Java with CREAM for Temporal Constraints

Malek Mouhoub and Mujtaba Istihad and Samira Sadaoui
Department of Computer Science
University of Regina
Regina SK, Canada, S4S 0A2

Abstract

Computational problems from many different application areas can be seen as temporal constraint-based problems. For example, scheduling, planning, computational linguistics and database design applications can all be seen in this way. In this paper we present a temporal constraint solver based on the Java Cream constraint library for managing problems involving numeric and symbolic temporal information. The solving system comes with a graphic user interface that allows the user to input the temporal information of a given problem and to check the consistency of these constraints in an interactive manner.

1 Introduction

Computational problems from many different application areas can be seen as constraint-based problems. For example, the problems of scheduling a collection of tasks, or interpreting a visual image, or laying out a silicon chip, can all be seen in this way. Because of the importance of these problems in so many different fields, a wide variety of techniques and programming languages from artificial intelligence, computational logic, operations research and discrete mathematics are being developed to tackle problems of this kind.

Constraint-based systems became more popular after they were incorporated into constraint programming languages. Since constraints provide a declarative way of representing sets of data, predicate logic was the natural choice as a framework for expressing data descriptions. This motivates the scientific community to develop “PROLOG-like” constraint programming languages well known as constraint logic programming languages. Some of these languages are PROLOG IV[5], GNU PROLOG, CHIP and ECLiPSe[3]. In the recent years other constraint programming languages have been inspired by other programming paradigms such as object oriented languages (ILOG Solver[2] and CHIP++ based C++), functional (Screamer based on Lisp) or multi-paradigm languages (Oz[6] combining the salient features of object oriented, functional, logic and concurrent programming into a single coherent design).

In this paper we present a temporal constraint solving system based on the Java Cream constraint solver. The proposed system allows the management of numeric and symbolic temporal constraints in an efficient way. This is very relevant for many real world applications such as scheduling, planning, molecular biology, database design and GIS systems.

The rest of the paper is structured as follows. Section 2 describes the fundamental concepts of our model TemPro[4] which is the core of our solving system. In Section 4 we present our solving system based on Java Cream. Concluding remarks and possible perspectives are finally listed in Section 5.

2 Managing Temporal Constraints: The Model TEMPRO

One important issue when dealing with problems involving temporal information is the ability to manage both the symbolic and numeric aspects of time. This motivates us to develop the model TemPro[4], extending the Interval Algebra defined by Allen[1] in order to handle numeric constraints. TemPro transforms any problem under qualitative and quantitative constraints into a binary CSP where constraints are disjunctions of Allen primitives[1] and variables, representing temporal events, are defined on domains of time intervals. We call this later a Temporal Constraint Satisfaction Problem (TCSP). Each event domain (called also temporal window) contains the Set Of Possible Occurrences (SOPO) of numeric intervals the corresponding event can take. The SOPO is the numeric constraint of the event. It is expressed by the fourfold \[ \text{earliest_start, latest_end, duration, step} \]
where: earliest_start is the earliest start time of the
event, latest_end is the latest end time of the event, duration is the duration of the event and step is the discretization step corresponding to the number of time units between the start time of two adjacent intervals belonging to the event domain. To illustrate the different components of the model TemPro, let us consider the following scheduling problem.

**Example 1**

The production of two items A and B requires three mono processor machines M1, M2 and M3. Each of the two items can be produced using two different ways depending on the order in which the machines are used. The process time of each machine is variable and depends on the task to be processed. The following lists the different ways to produce each of the two items (the process time for each machine is mentioned in brackets):

**item A:** M2(3), M1(3), M3(6) or M2(3), M3(6), M1(3)

**item B:** M2(2), M1(5), M2(2), M3(7) or M2(2), M3(7), M2(2), M1(5)

The goal here is to find a possible schedule of the different machines to produce the two items while respecting all the constraints of the problem. We also assume that items A and B should be produced within 25 and 30 units of time respectively. Figure 1 illustrates the graph representation of the TCSP corresponding the the scheduling problem. A temporal event corresponds here to the contribution of a given machine to produce a certain item.

**3 CREAM for Managing Temporal Constraints**

The solving system we propose was developed by coding in JAVA (using the JAVA CREAM library) our model TemPro and the corresponding resolution methods we have discussed in Section 2. We have used the NetBeans environment and to include Cream class li-
library, we mounted `cream.jar` file, which is available at the location given in section 3. Our solving system offers the following features:

- **Managing numeric and symbolic temporal information.** Using the graphic user interface, the user can input a list of symbolic and numeric constraints corresponding to a given temporal problem. The system will then display the temporal constraint graph corresponding to the problem. The user may then ask to check the consistency of the problem. The system will then display the result of the different steps of the consistency check (arc consistency, path consistency and backtrack search). One solution, all solutions or a preferred one (for example a solution minimizing the total process time of the different events) can be displayed depending on the choice given by the user.

- **Solving Temporal Constraint Problems in Real Time.** In this case the user can input a temporal constraint problem (same as shown in the previous case) and a given allocated time. The allocated time is the maximum time allowed to find a solution. If a complete solution (satisfying all the constraints of the problem) cannot be found during this time, a partial solution (minimizing the number of violated constraints) will be returned.

- **Handling Temporal Constraints in a Dynamic Environment.** The user can use the solving system in a dynamic manner. He can, for example, input a list of constraints corresponding to a given problem and run the consistency check. The user can then add other constraint during or after the resolution process. The solving system will then process the new constraints in an incremental way (without the need to resolve the entire problem from scratch). Also, if a given problem is not consistent, the user can choose the constraint to be retracted from the problem. The system will then re-check the consistency of the problem in an incremental way (only the variables affected by this change will be considered).

Figure 2 illustrates the graphic user interface of our solving system. The temporal problem to input consists here of a number of persons events and their start and end times. The relative constraints between the events of the different persons are also known and our goal is to find a consistent scenario (assignment of numeric intervals to temporal events) that satisfies all the constraints. For each variable, inputs are given as the earliest start time, the latest start time and the duration.

4 Conclusion

In this paper we have presented a temporal solving system capable of solving problems under numeric and symbolic temporal constraints. Through a friendly graphic user interface, the user can input a given temporal constraint problem in the form of a list of variables defined on domains of temporal intervals and a list of temporal constraints between the variables. The system will then visualize the initial problem (as a constraint graph) and the different steps of the resolution process. The user can also ask to solve a problem within a given deadline and in an evolutive manner. In this later case, the user can add or remove constraints during the resolution process and see the effect of this change in an incremental way.

One perspective of our work is to extend the system in order to handle conditional temporal constraint problems. Conditional temporal problems are temporal problems containing temporal variables whose existence depends on the values chosen for other variables. In this case we have to maintain the consistency of the temporal problem anytime new temporal variables are added.

References